CASSAFORTE A DOPPIA CHIAVE – DI MAURO ENRICO MARIA 0612704486 – GRUPPO 29

TRACCIA 3

Si realizzi un controllore per apertura di una cassaforte. Il controllore è basato sul riconoscimento di due sequenze di 4 bit generate da due chiavi a disposizione di personale di sicurezza. Il funzionamento è il seguente. Il primo utente deve inserire la sua chiave in una serratura. Il controllore di apertura rileva la chiave e si pone in attesa di ricevere una sequenza generata da un componente esterno (NOTA: ciò significa che la sequenza è in ingresso al controllore di apertura). Il secondo utente deve fare la stessa cosa con la sua chiave, ed il controllore di apertura si comporta in maniera analoga. Se viene riconosciuta una sequenza e, entro un tempo limite (per esempio, 10 cicli di clock), non si riconosce (per qualunque motivo) la seconda il controllore di apertura si riporta allo stato inziale e la procedura deve ripartire. La cassaforte si apre se e solo se le due chiavi sono inserite, e le due sequenze riconosciute entro il tempo limite. L'ordine delle serrature non è importante, lo studente può scegliere le sequenze di 4 bit.

FUNZIONAMENTO & SCELTE PROGETTUALI

Il principio di funzionamento della nostra cassaforte consiste nell' inserimento di una prima chiave che deve essere riconosciuta da uno dei 2 lettori, non vi è vincolo su quale chiave debba essere riconosciuta per prima poiché abbiamo considerato che le chiavi possono essere tranquillamente invertite senza generare errori. Una volta riconosciuta la prima chiave si deve procedere nell'arco di tempo di 10 cicli di clock all'inserimento e riconoscimento della seconda chiave. Se entrambe le chiavi sono state riconosciute con successo la serratura della porta si sbloccherà e ci permetterà di aprire la cassaforte. Per come abbiamo deciso di implementare la macchina è anche possibile inserire contemporaneamente entrambe le chiavi corrette in due serrature diverse per aprire la cassaforte. Nel momento in cui la porta verrà aperta non sarà possibile richiuderla definitivamente a meno che non vengano estratte entrambe le chiavi e la porta non venga accostata in modo da far attivare un sensore che ne causerà il blocco. Se entro i 10 cicli di clock non viene registrato l'inserimento corretto della seconda chiave si entrerà in uno stato in cui sarà possibile unicamente reiterare il processo solo se verranno estratte entrambe le chiavi. Il display a sette segmenti aiuterà l'utente nella procedura di apertura della cassaforte segnalando eventuali errori.

Tutti gli ingressi e le uscite utilizzati sono considerati come sensori e attuatori a livelli.

Le chiavi vengono considerate come dispositivi elettronici esterni che generano una sequenza di 4 bit.

Per la realizzazione della nostra cassaforte è stato scelto un approccio TOP-DOWN, dividere un problema in sotto-problemi più semplici, cioè MODULARE (il diagramma a blocchi consisterà proprio nel RTL Schematic).

Abbiamo deciso di scomporre il controllore (TOP MODULE) in 4 moduli:

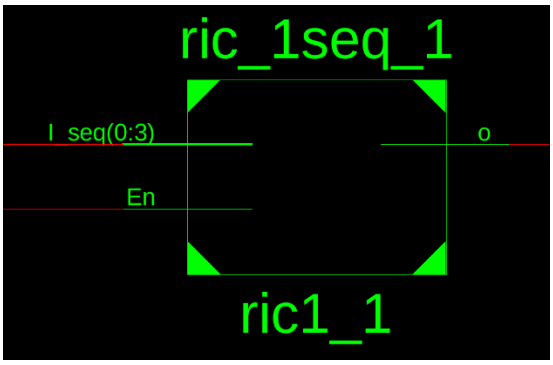
- RICONOSCITORE DI UNA SEQUENZA
- DISPLAY A SETTE SEGMENTI
- CONTATORE DI N CICLI DI CLOCK
- GESTORE DEGLI STATI DELLA CASSAFORTE

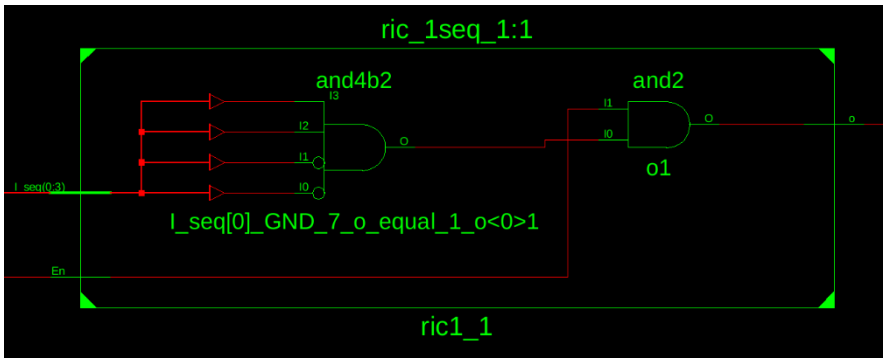
sezioni aggiuntive:

DESIGN SUMMARY REPORT

APPROFONDIMENTI

➢ *RICONOSCITORE DI UNA SEQUENZA* *(torna su)*





Entity del modulo

Esempio di RTL-Schematic del componente con seq="0011"

Questo modulo ci permette di riconoscere una determinata sequenza di bit data in input. Se la sequenza viene riconosciuta l'uscita sarà pari ad 1, 0 altrimenti.

I suoi ingressi sono:

- En che sarebbe l'enable. Ci permette di abilitare o disabilitare il componente. 1 abilita, 0 altrimenti
- I_seq che sarebbe la sequenza di 4 bit da riconoscere

L'uscita è:

- o che sarebbe il segnale che mi segnala con 1 se la sequenza è stata riconosciuta, 0 altrimenti

Codice: ric_1seq.vhd

```
19  ----------------------------------------------------------------------
20  library IEEE;
21  use IEEE.STD_LOGIC_1164.ALL;
22
23  -- Uncomment the following library declaration if using
24  -- arithmetic functions with Signed or Unsigned values
25  --use IEEE.NUMERIC_STD.ALL;
26
27  -- Uncomment the following library declaration if instantiating
28  -- any Xilinx primitives in this code.
29  --library UNISIM;
30  --use UNISIM.VComponents.all;
31
32  entity ric_1seq is
33  Generic(
34     seq : std_logic_vector(0 to 3) :="0000"
35  );
36  Port(
37     En : in std_logic;
38     I_seq : in std_logic_vector(0 to 3);
39
40     o : out std_logic
41  );
42  end ric_1seq;
43
44  architecture Behavioral of ric_1seq is
45
46  begin
47
48  process(En,I_seq)
49  begin
50     if seq=I_seq and En='1' then
51        o<='1';
52     else
53        o<='0';
54     end if;
55
56  end process;
57
58
59  end Behavioral;
60
61
```

Abbiamo deciso di utilizzare un parametro generic in modo tale da poter implementare questo componente più volte all' interno del Top Module e quindi renderlo un modulo di riconoscitore di sequenza generico a 4 bit. Se il generic non viene definito durante l'istanziazione del componente verrà presa in considerazione la sequenza di default "0000".

Come si può notare abbiamo utilizzato un approccio procedurale (Behavioral) che si occupa di controllare l'uguaglianza della sequenza in ingresso (I_seq) con quella da riconoscere (seq) ed inoltre è stato aggiunto un enable (En) che permette l'abilitazione o meno di questo modulo. Dato che ad ogni variazione dell'enable e della sequenza data in ingresso avverranno variazioni sull'uscita (o) questi 2 ingressi dovranno far parte della sensitivity list del process.

Testbench: test_ric1.vhd

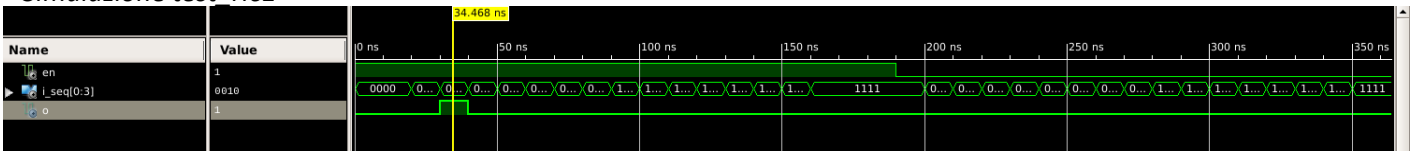
```
27  ----------------------------------------------------------------------
28  LIBRARY ieee;
29  USE ieee.std_logic_1164.ALL;
30  USE ieee.std_logic_arith.ALL;
31
32  -- Uncomment the following library declaration if using
33  -- arithmetic functions with Signed or Unsigned values
34  --USE ieee.numeric_std.ALL;
35
36  ENTITY test_ric1 IS
37  END test_ric1;
38
39  ARCHITECTURE behavior OF test_ric1 IS
40
41      -- Component Declaration for the Unit Under Test (UUT)
42
43      COMPONENT ric_1seq
44      GENERIC(
45        seq : std_logic_vector(0 to 3)
46      );
47      PORT(
48           En : IN  std_logic;
49           I_seq : IN  std_logic_vector(0 to 3);
50           o : OUT  std_logic
51          );
52      END COMPONENT;
53
54
55     --Inputs
56     signal En : std_logic := '0';
57     signal I_seq : std_logic_vector(0 to 3) := (others => '0');
58
59     --Outputs
60     signal o : std_logic;
61     -- No clocks detected in port list. Replace <clock> below with
62     -- appropriate port name
63
64
65  BEGIN
66
67     -- Instantiate the Unit Under Test (UUT)
68     uut: ric_1seq
69     GENERIC MAP(seq=>"0010") --sequenza da riconoscere
70     PORT MAP (
71            En => En,
72            I_seq => I_seq,
73            o => o
74          );
75
76
77
78     -- Stimulus process
79     stim_proc: process
80     begin
81
82       En<='1';
83       wait for 10 ns;
84
85       for i in 0 to 15 loop
86           I_seq<=conv_std_logic_vector(i,4);
87           wait for 10ns;
88       end loop;
89
90       wait for 20 ns;
91       En<='0';
92       wait for 10ns;
93
94       for i in 0 to 15 loop
95           I_seq<=conv_std_logic_vector(i,4);
96           wait for 10ns;
97       end loop;
98
99       wait;
100    end process;
101
102 END;
```

Come si può notare alla riga 69 abbiamo definito come sequenza da riconoscere “0010”.

Il funzionamento di questa testbench prevede lo scorrimento di tutte le possibili combinazioni una volta con l’enable pari a 1 ed una volta con enable pari a 0.

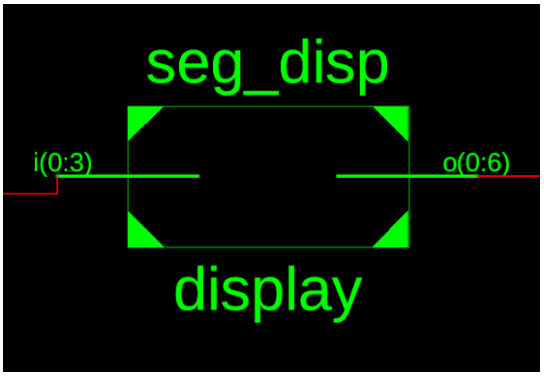
Per fare ciò abbiamo utilizzato un for che va da 0 a 2^n_bit-1 con n_bit pari a 4. All’interno del for abbiamo assegnato questo numero, convertito in std_logic_vector su 4 bit, all’ingresso I_seq.

Simulazione test_ric1

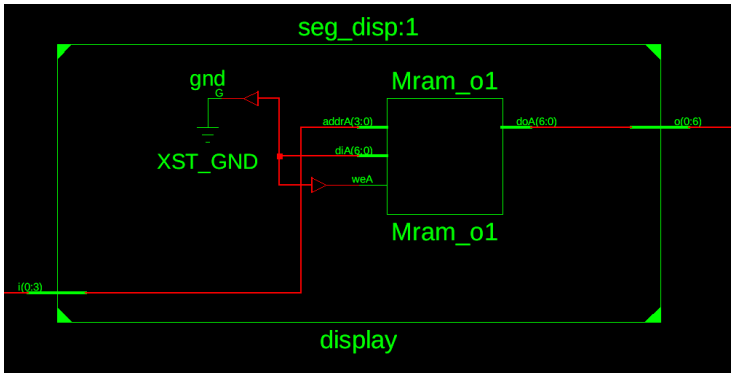


Come si può ben notare dalla simulazione solamente quando la sequenza in ingresso è proprio “0010” ed En=1 allora avremo o=1, 0 altrimenti.

➢ *DISPLAY A SETTE SEGMENTI* *(torna su)*



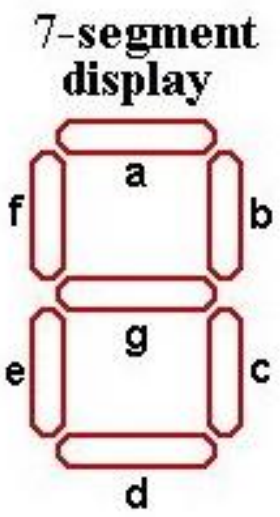
Entity del modulo



RTL-Schematic del componente

Digit	Display	gfedcba	abcdefg	a	b	c	d	e	f	g
0	0	0×3F	0×7E	on	on	on	on	on	on	off
1	1	0×06	0×30	off	on	on	off	off	off	off
2	2	0×5B	0×6D	on	on	off	on	on	off	on
3	3	0×4F	0×79	on	on	on	on	off	off	on
4	4	0×66	0×33	off	on	on	off	off	on	on
5	5	0×6D	0×5B	on	off	on	on	off	on	on
6	6	0×7D	0×5F	on	off	on	on	on	on	on
7	7	0×07	0×70	on	on	on	off	off	off	off
8	8	0×7F	0×7F	on	on	on	on	on	on	on
9	9	0×6F	0×7B	on	on	on	on	off	on	on
A	A	0×77	0×77	on	on	on	off	on	on	on
b	b	0×7C	0×1F	off	off	on	on	on	on	on
C	C	0×39	0×4E	on	off	off	on	on	on	off
d	d	0×5E	0×3D	off	on	on	on	on	off	on
E	E	0×79	0×4F	on	off	off	on	on	on	on
F	F	0×71	0×47	on	off	off	off	on	on	on

Tabella delle combinazioni



Questo modulo ci permette di convertire un numero codificato in bcd rappresentante i numeri da 0 a 9 e le lettere da A ad F (numeri che vanno da 10 a 15) in modo da visualizzarli sul display.

I suoi ingressi sono:

- i che è un unsigned da 4 bit che rappresenta il valore bcd in ingresso da visualizzare sul display

Le sue uscite sono:

- o che è il vettore da 7 bit che rappresenta i segmenti del display nell'ordine "abcdefg" che devono accendersi per visualizzare il valore bcd dato in ingresso

Codice: ric_1seq.vhd

```vhdl
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.std_logic_arith.ALL;

-- Uncomment the following library declaration if using
-- arithmetic functions with Signed or Unsigned values
--use IEEE.NUMERIC_STD.ALL;

-- Uncomment the following library declaration if instantiating
-- any Xilinx primitives in this code.
--library UNISIM;
--use UNISIM.VComponents.all;

entity seg_disp is
   port
   (
      i :in unsigned(0 to 3);
      o :out STD_Logic_vector(0 to 6)
   );
end seg_disp;

architecture Dataflow of seg_disp is

begin

with i select
   o<=   "1111110" when "0000",
         "0110000" when "0001",
         "1101101" when "0010",
         "1111001" when "0011",
         "0110011" when "0100",
         "1011011" when "0101",
         "1011111" when "0110",
         "1110000" when "0111",
         "1111111" when "1000",
         "1111011" when "1001",
         "1110111" when "1010",
         "0011111" when "1011",
         "1001110" when "1100",
         "0111101" when "1101",
         "1001111" when "1110",
         "1000111" when "1111",
         "-------" when others;

end Dataflow;
```

Per la gestione di questo componente abbiamo utilizzato un'architecture di tipo Dataflow.

Per rappresentare la tabella per convertire un numero da bcd a codifica a 7 segmenti abbiamo scelto di utilizzare un WITH SELECT che come si può notare nella sintesi del componente è rappresentato da una RAM.

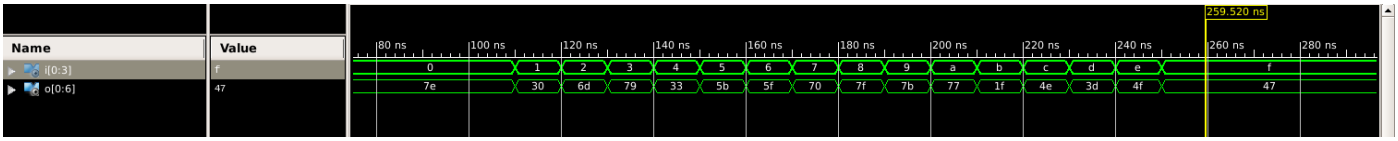
Testbench: test_seg_disp.vhd

```vhdl
LIBRARY ieee;
USE ieee.std_logic_1164.ALL;
use IEEE.std_logic_unsigned.ALL;
use IEEE.std_logic_arith.ALL;
-- Uncomment the following library declaration if using
-- arithmetic functions with Signed or Unsigned values
--USE ieee.numeric_std.ALL;

ENTITY test_seg_disp IS
END test_seg_disp;

ARCHITECTURE behavior OF test_seg_disp IS

    -- Component Declaration for the Unit Under Test (UUT)

    COMPONENT seg_disp
    PORT(
         i : IN  unsigned(0 to 3);
         o : OUT std_logic_vector(0 to 6)
        );
    END COMPONENT;


   --Inputs
   signal i : unsigned(0 to 3) := (others => '0');

   --Outputs
   signal o : std_logic_vector(0 to 6);


BEGIN

   -- Instantiate the Unit Under Test (UUT)
   uut: seg_disp PORT MAP (
          i => i,
          o => o
        );


   -- Stimulus process
   stim_proc: process
   begin
      -- hold reset state for 100 ns.
      wait for 100 ns;


     for k in 0 to 15 loop
        i<=conv_unsigned(k,4);
        wait for 10 ns;
     end loop;

      wait;
   end process;

END;
```

Il funzionamento di questa testbench prevede lo scorrimento di tutte le possibili combinazioni.

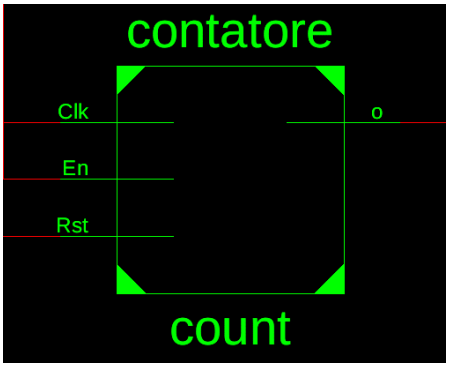
Per fare ciò abbiamo utilizzato un for che va da 0 a 2^n_bit-1 con n_bit pari a 4. All'interno del for abbiamo assegnato questo numero, convertito in unsigned su 4 bit, all'ingresso i.

Simulazione test_seg_disp

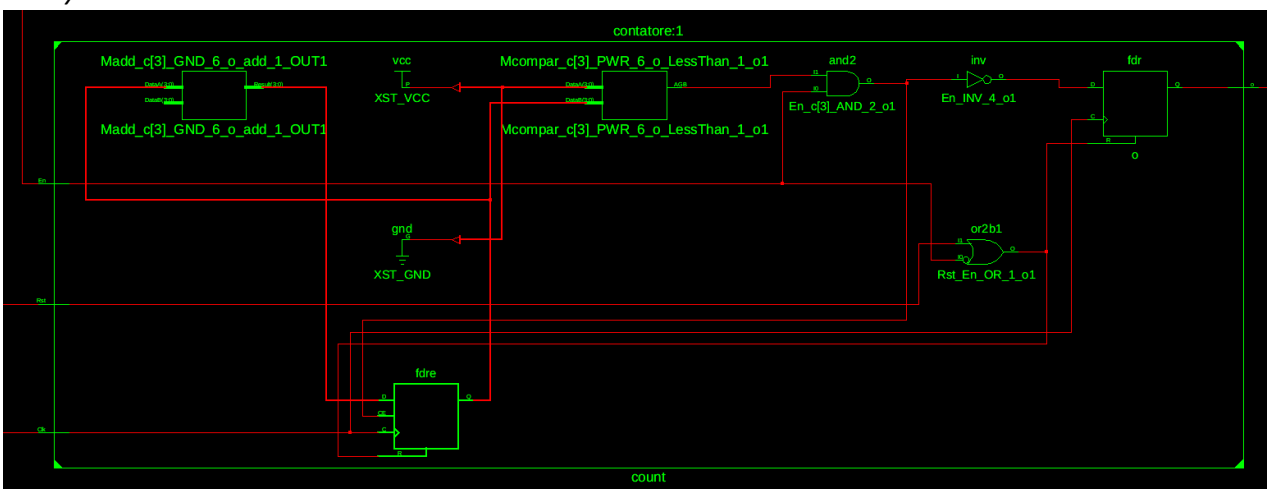


Per poter capire se il display funziona correttamente basta confrontare l'ingresso e l'uscita con radix HEX (visualizzazione esadecimale) con la tabella sopra riportata.

➢ *CONTATORE DI N CICLI DI CLOCK* *(torna su)*



Entity del modulo



Esempio di RTL-Schematic del componente con n=10

Questo modulo ci permette di creare un timer che conta n cicli di clock. Raggiunti gli n cicli di clock l'uscita va a 1, 0 altrimenti. Per abilitare il conteggio basta porre l'Enable pari a 1. Per resettare il conteggio e l'uscita basta porre l'enable pari a 0 oppure il rst pari a 1.

I suoi ingressi sono:

- En che sarebbe l'enable. Ci permette di abilitare o disabilitare (resettare) il conteggio. 1 abilita, 0 altrimenti
- rst che sarebbe il reset. Ci permette di resettare il conteggio e l'uscita
- clk che sarebbe il clock. Ci permette di sincronizzare l'intero modulo sequenziale

L'uscita è:

- o che sarebbe il segnale che mi segnala con 1 se sono scaduti gli n cicli di clock, 0 altrimenti

Codice: contatore.vhd

```
19  ----------------------------------------------------------------------
20  library IEEE;
21  use IEEE.STD_LOGIC_1164.ALL;
22
23  -- Uncomment the following library declaration if using
24  -- arithmetic functions with Signed or Unsigned values
25  --use IEEE.NUMERIC_STD.ALL;
26
27  -- Uncomment the following library declaration if instantiating
28  -- any Xilinx primitives in this code.
29  --library UNISIM;
30  --use UNISIM.VComponents.all;
31
32  entity contatore is
33  Generic(
34     n : integer := 10
35  );
36  Port(
37     Clk,Rst : in std_logic;
38     En : in std_logic;
39
40     o : out std_logic
41
42  );
43  end contatore;
44
45  architecture Behavioral of contatore is
46     signal c : integer range 0 to n;
47  begin
48
49  process(clk)
50  begin
51     if rising_edge(clk) then
52        if Rst='1' or En='0' then
53           c<=0;
54           o<='0';
55        elsif En='1' and c<n then
56           o<='0';
57           c<=c+1;
58        else
59           c<=c;
60           o<='1';
61        end if;
62     end if;
63  end process;
64
65  end Behavioral;
```

Abbiamo deciso di utilizzare un parametro generic in modo tale da realizzare un modulo che possa contare n cicli di clock, scegliendo n al momento della definizione del componente. Se il generic non viene definito durante l'istanziazione del componente verrà preso in considerazione n pari a 10, come si può intuire dal codice.

Come si può notare abbiamo utilizzato un approccio procedurale (Behavioural), in cui il segnale contatore interno all'architecture si occupa di contare i cicli del clock e portare l'uscita a 1 nel momento in cui avrà raggiunto il numero di cicli scelto.

Abbiamo una variabile c contatore che memorizza il conteggio dei cicli di clock, questa variabile può assumere valori che vanno da 0 ad n. Ad ogni rising_edge il valore di c viene incrementato di 1 e l'uscita viene mantenuta a 0 solo se l'enable è pari a 1, il rst è pari a 0 e il contatore non ha ancora raggiunto n. L'uscita verrà portata a 1 solo se l'enable è pari a 1 e il contatore ha raggiunto il numero di cicli desiderato. Se il rst è pari a 1 o l'enable è pari a 0, il contatore e l'uscita verranno portate a 0.

Come notiamo nella sensitivity list del process abbiamo solamente il clock poiché vi saranno aggiornamenti del contatore e variazioni di uscita solamente sul fronte di salita del clock.

Testbench: test_cont.vhd

```
27  ----------------------------------------------------------------
28  LIBRARY ieee;
29  USE ieee.std_logic_1164.ALL;
30
31  -- Uncomment the following library declaration if using
32  -- arithmetic functions with Signed or Unsigned values
33  --USE ieee.numeric_std.ALL;
34
35  ENTITY test_cont IS
36  END test_cont;
37
38  ARCHITECTURE behavior OF test_cont IS
39
40      -- Component Declaration for the Unit Under Test (UUT)
41
42      COMPONENT contatore
43      GENERIC(
44           n : integer
45      );
46      PORT(
47           Clk : IN  std_logic;
48           Rst : IN  std_logic;
49           En : IN  std_logic;
50           o : OUT  std_logic
51          );
52      END COMPONENT;
53
54
55     --Inputs
56     signal Clk : std_logic := '0';
57     signal Rst : std_logic := '0';
58     signal En : std_logic := '0';
59
60     --Outputs
61     signal o : std_logic;
62
63     -- Clock period definitions
64     constant Clk_period : time := 10 ns;
65
66  BEGIN
67
68     -- Instantiate the Unit Under Test (UUT)
69     uut: contatore
70        GENERIC MAP(
71           n=>10
72        )
73        PORT MAP (
74            Clk => Clk,
75            Rst => Rst,
76            En => En,
77            o => o
78           );
79
80     -- Clock process definitions
81     Clk_process :process
82     begin
83        Clk <= '0';
84        wait for Clk_period/2;
85        Clk <= '1';
86        wait for Clk_period/2;
87     end process;
88
89
90     -- Stimulus process
91     stim_proc: process
92     begin
93        -- hold reset state for 100 ns.
94        rst<='1';
95        wait for 100 ns;
96        rst<='0';
97        wait for Clk_period*10;
98
99
100       -- insert stimulus here
101       rst<='0';
102       en<='0';
103       wait for 120 ns;
104       rst<='0'; --caso in cui il contatore è abilitato
105       en<='1';
106       wait for 120 ns;
107       rst<='1';
108       en<='0';
109       wait for 120 ns;
110       rst<='1';
111       en<='1';
112
113       wait;
114
115    end process;
116
117 END;
```

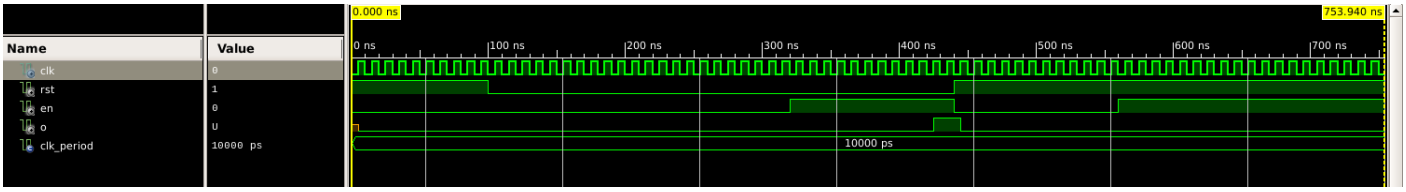
Come si può notare alla riga 71 abbiamo definito come generic il numero di cicli che il clock deve contare.

Il funzionamento di questa testbench prevede il test di tutte le combinazioni possibili date reset ed enable.

Dopo ogni test abbiamo atteso 120 nanosecondi per assicurarci che l'uscita non venisse portata ad 1 anche in condizioni non ammissibili, ma solamente quando avremo rst=0 ed en=1, proprio come indicato da commento.

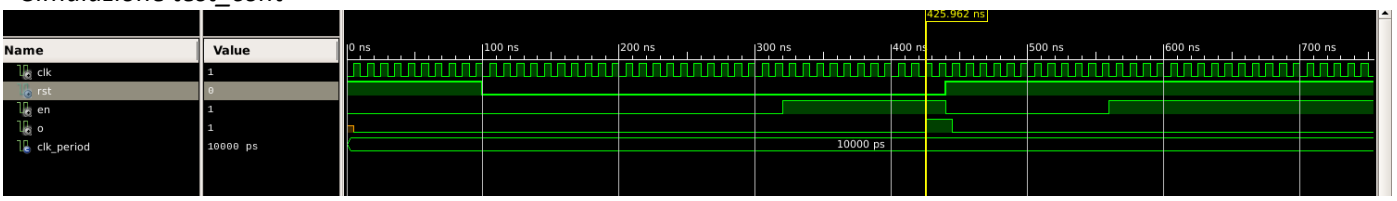
Essendo questo un modulo formato da una rete sequenziale sincrona prima di procedere con i vari test degli ingressi abbiamo resettato la macchina come si nota nelle righe 94-97.

Simulazione test_cont



Come si può notare all'inizio della simulazione l'uscita non ha un valore definito, proprio per questo motivo resettiamo il tutto prima di poter procedere con l'effettivo test a partire dai 200 nanosecondi.

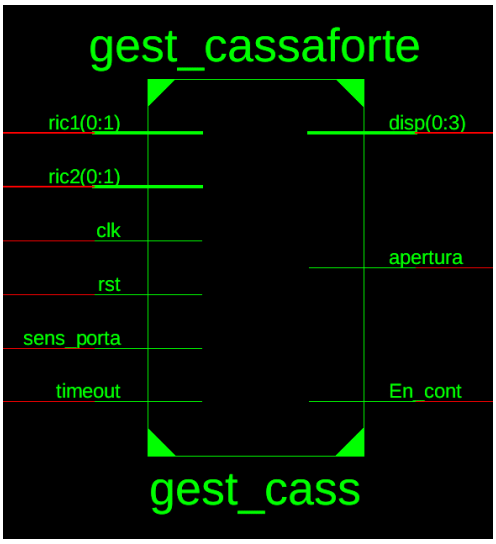
Simulazione test_cont



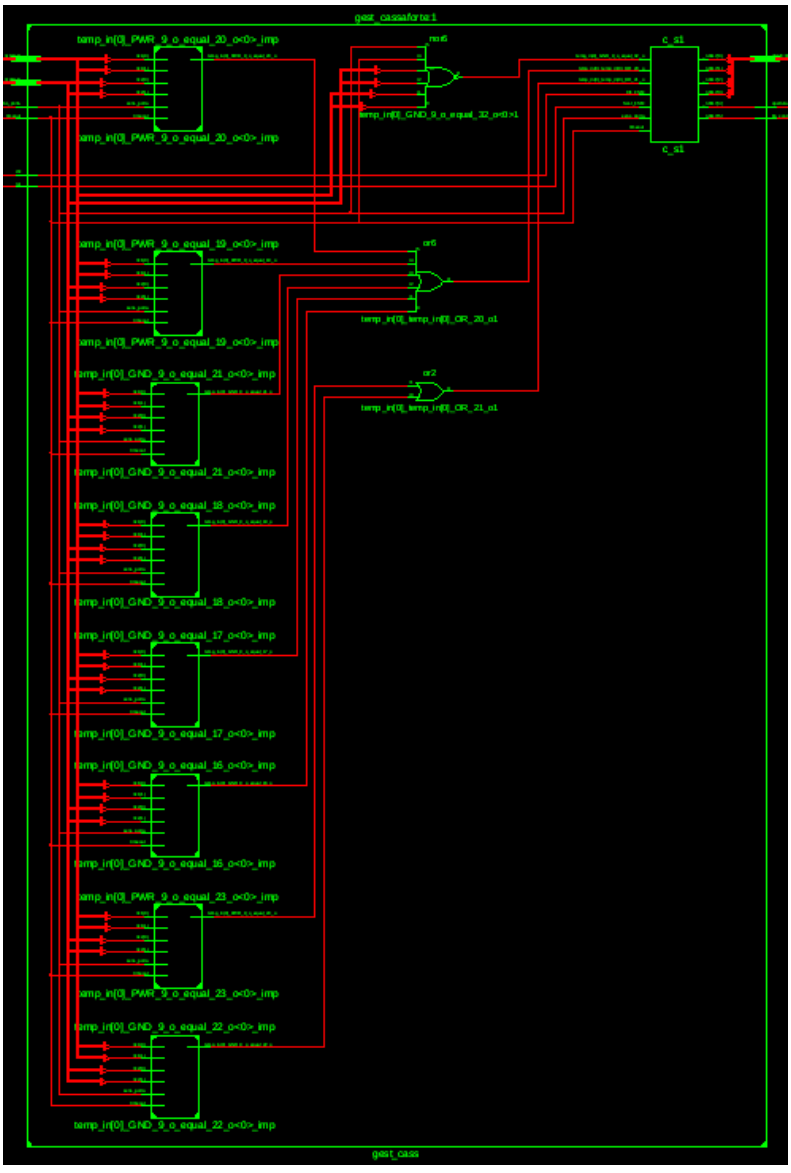
Come si può notare dai valori di ingressi ed uscita, solo dopo 10 cicli di clock a partire dal momento in cui rst=0 e en=1 si otterrà o=1, 0 altrimenti.

➢ *GESTORE DEGLI STATI DELLA CASSAFORTE*

(torna su)



Entity del modulo



RTL-Schematic del componente

Questo modulo ci permette di gestire il funzionamento della nostra cassaforte.

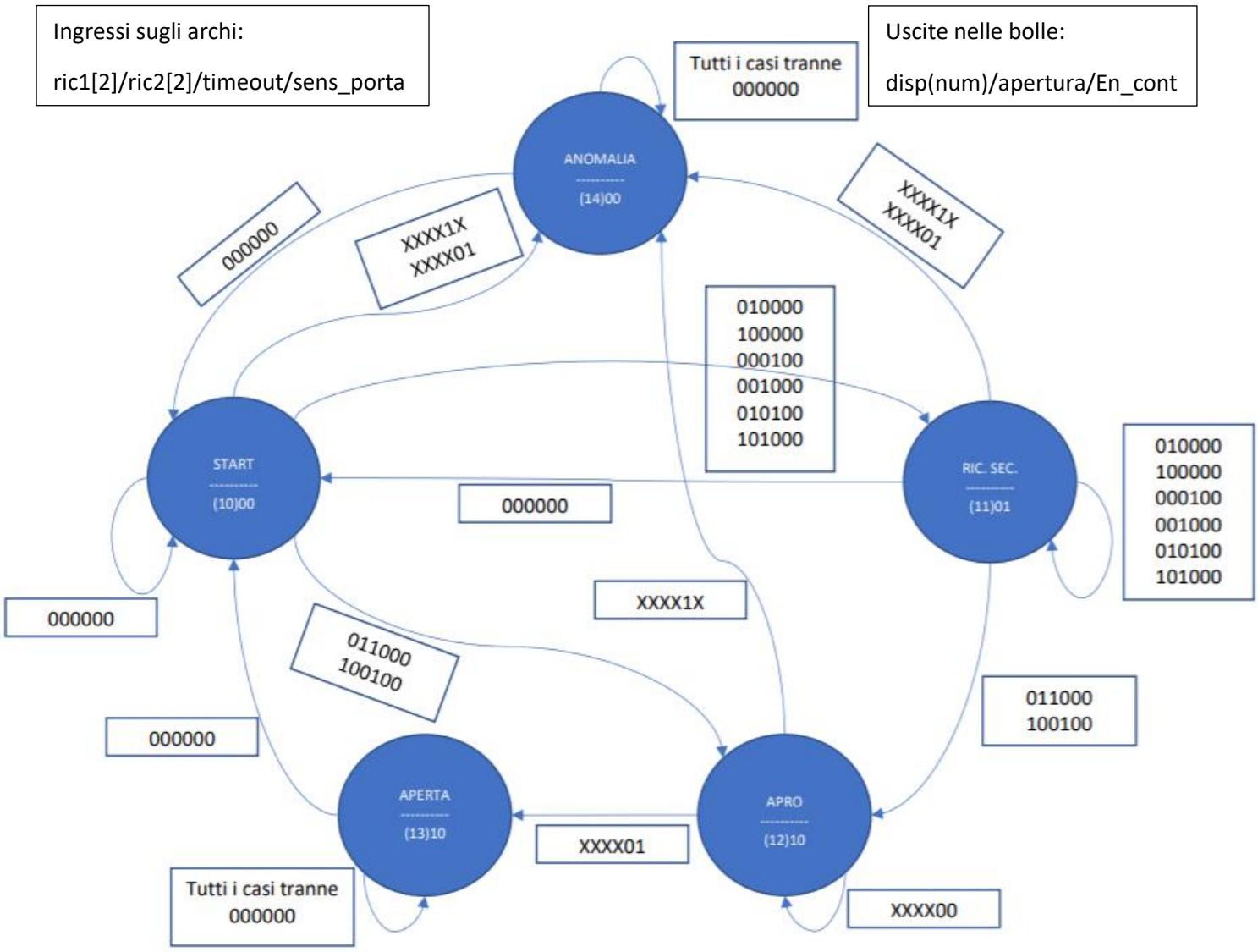
I suoi ingressi sono:

- ric1 e ric2 che sarebbero vettori di 2 bit per ciascuna chiave che indicano: 00 se nessuna sequenza è stata riconosciuta o nessuna chiave è stata inserita, 01 se è stata riconosciuta la prima sequenza, 10 se è stata riconosciuta la seconda sequenza, **11 sequenza non ammissibile, non è stata presa in considerazione per questo automa** poiché significherebbe il riconoscimento di 2 sequenze con l'inserimento di un'unica chiave
- sens_porta che sarebbe il sensore che indica se la porta è chiusa se vale 0, 1 altrimenti
- timeout che sarebbe il segnale che indica la fine del tempo ammissibile per il riconoscimento della seconda chiave, vale 1 se il tempo è scaduto, 0 altrimenti
- clk che è il segnale rappresentante il clock su cui si sincronizza tutto il modulo
- rst che è il segnale che resetta il modulo

Le sue uscite sono:

- apertura che sarebbe la serratura, è bloccata se vale 0, 1 altrimenti
- En_cont che sarebbe il segnale che gestisce il timer, lo abilita se vale 1, 0 altrimenti
- disp che sarebbe un unsigned rappresentante il numero da visualizzare su un eventuale display collegato a questo modulo

AUTOMA DI MOORE



Basandoci sugli stati visualizzati nell'automa il suo funzionamento è il seguente:

- START (stato iniziale): lettera sul display=A (10), serratura bloccata, contatore disabilitato. È lo stato in cui permaniamo fino a quando non viene inserita almeno una chiave corretta. Se il sensore della porta rileva erroneamente un'apertura della porta o timeout=1 la cassaforte si porterà sullo stato di ANOMALIA poiché ciò non è ammissibile in questo stato. Se una sola delle 2 chiavi viene riconosciuta correttamente e l'altra no ci portiamo sullo stato di riconoscimento della seconda chiave (RIC. SEC.). Qualora venissero riconosciute contemporaneamente entrambe le chiavi corrette ci porteremmo sullo stato che ci abilita l'apertura della cassaforte (APRO)
- RIC. SEC.: lettera sul display=b (11), serratura bloccata, contatore abilitato. È lo stato in cui si attende il riconoscimento della seconda chiave fin quando il timeout non risulta pari ad 1 o si verifica un caso di anomalia. Nel caso in cui timeout=1 o erroneamente risulta la porta aperta tramite il sensore si va in stato di ANOMALIA. Se viene riconosciuta la seconda chiave fin quando timeout=0 allora si va sullo stato che ci abilita l'apertura della cassaforte (APRO). Se nessuna chiave viene più riconosciuta si ritorna nello stato iniziale (START)
- ANOMALIA: lettera sul display=E (14), serratura bloccata, contatore disabilitato. È lo stato che indica un effettivo malfunzionamento o errore dovuto al non riconoscimento della seconda chiave dopo che timeout è diventato 1. È possibile tornare allo stato iniziale solo se tutte le chiavi vengono disinserite, la porta risulta chiusa ed il timeout=0 o la macchina viene resettata, si permane in questo stato altrimenti
- APRO: lettera sul display=C (12), serratura sbloccata, contatore disabilitato. È lo stato in cui la porta risulta chiusa, ma la serratura è sbloccata. Se la porta viene aperta si passa allo stato di apertura effettiva (APERTA). Se erroneamente il timeout dovesse diventare pari ad 1 si andrebbe in stato di ANOMALIA
- APERTA: lettera sul display=d (13), serratura sbloccata, contatore disabilitato. È lo stato in cui la porta risulta aperta correttamente. Solo nel momento in cui la porta venga chiusa con entrambe le chiavi estratte si tornerà allo stato iniziale (START)

È da tener conto che:

Il timeout non potrebbe mai diventare pari ad 1 tranne che nello stato RIC. SEC. in cui viene abilitato il timer, ma è stato ugualmente gestito, infatti in START e APRO in questo caso si va in ANOMALIA mentre in APERTO permaniamo nello stato poiché non è possibile andare in ANOMALIA siccome la porta è aperta e verrebbe chiusa in modo anomalo.

Negli stati START o RIC.SEC. si suppone che il sensore della porta non potrebbe mai rilevare la porta aperta poiché è serrata e quindi ciò provocherebbe il passaggio allo stato ANOMALIA.

Nello stato APRO rimarrò all'infinito finché non verrà aperta la porta.

Codice: gest_cassaforte.vhd

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.std_logic_unsigned.ALL;
use IEEE.std_logic_arith.ALL;
-- Uncomment the following library declaration if using
-- arithmetic functions with Signed or Unsigned values
--use IEEE.NUMERIC_STD.ALL;

-- Uncomment the following library declaration if instantiating
-- any Xilinx primitives in this code.
--library UNISIM;
--use UNISIM.VComponents.all;

entity gest_cassaforte is
Port(
      clk,rst : in std_logic;
      ric1 : in std_logic_vector(0 to 1);
      ric2 : in std_logic_vector(0 to 1);
      sens_porta: in std_logic;
      timeout: in std_logic;

      apertura : out std_logic;
      disp: out unsigned(0 to 3);
      En_cont: out std_logic
);
end gest_cassaforte;

architecture Behavioral of gest_cassaforte is



   type stato is(start,ric_sec,anomalia,apro,aperta);
   signal c_s,n_s :stato;

   signal temp_in : std_logic_vector(0 to 5);   --vettore temporaneo che concatena gli ingressi in modo da facilitare la scrittura delle transizioni di stato
   signal temp_out : std_logic_vector(0 to 1);  --vettore temporaneo in cui vengono inserite le uscite per poi essere assegnate alle uscite del modulo

begin

   temp_in<=ric1 & ric2 & timeout & sens_porta; --concatenazione ingressi

   process(clk)   --processo che memorizza lo stato corrente e gestisce il reset
   begin

      if rising_edge(clk) then
         if rst='1' then
            c_s<=start;
         else
            c_s<=n_s;
         end if;
      end if;

   end process;

   process(c_s,temp_in)    --processo che produce il next state e le uscite per Moore
   begin
      case c_s is
         when start=>
                  disp<=conv_unsigned(10,4);
                  temp_out<="00" ;

                  if temp_in="000000" then
                     n_s<=start;
                  elsif temp_in="010000" or temp_in="010100" or temp_in="000100" or temp_in="100000" or temp_in="101000" or temp_in="001000" then
                     n_s<=ric_sec;
                  elsif temp_in="011000" or temp_in="100100" then
                     n_s<=apro;
                  else
                     n_s<=anomalia;
                  end if;
         when ric_sec=>
                  disp<=conv_unsigned(11,4);
                  temp_out<="01";

                  if temp_in="010000" or temp_in="010100" or temp_in="000100" or temp_in="100000" or temp_in="101000" or temp_in="001000" then
                     n_s<=ric_sec;
                  elsif temp_in="011000" or temp_in="100100" then
                     n_s<=apro;
                  elsif temp_in="000000" then
                     n_s<=start;
                  else
                    n_s<=anomalia;
                  end if;
         when anomalia=>
                  disp<=conv_unsigned(14,4);
                  temp_out<="00";

                  if temp_in="000000"  then
                     n_s<=start;
                  else
                     n_s<=anomalia;
                  end if;
         when apro=>
                  disp<=conv_unsigned(12,4);
                  temp_out<="10";

                  if temp_in(5)='1' and temp_in(4)='0' then
                     n_s<=aperta;
                  elsif temp_in(5)='0' and temp_in(4)='0' then
                     n_s<=apro;
                  else
                     n_s<=anomalia;
                  end if;
         when aperta=>
                  disp<=conv_unsigned(13,4);
                  temp_out<="10";

                  if temp_in="000000" then
                     n_s<=start;
                  else
                     n_s<=aperta;
                  end if;
      end case;
   end process;
--assegnazione delle uscite temporanee e quelle effettive del modulo
   En_cont<=temp_out(1);
   apertura<=temp_out(0);

end Behavioral;
```

Per la gestione dell'automa abbiamo utilizzato il template di Moore e quindi una vista Behavioral.

Per la gestione di tale template abbiamo bisogno sicuramente di 2 process: il primo che memorizza lo stato corrente, il secondo che determina il next state e le uscite.

Nella sensitivity list del primo process andrà solamente il clock poiché sia reset che la determinazione del current state sono sincroni sul fronte di salita del clock.

Moore

$$o_t = \omega'(s_t)$$

$$s_{t+1} = \sigma(i_t, s_t)$$

Nella sensitivity list del secondo process avremo sia il current state sia gli ingressi e siccome la determinazione dell'uscita dipende dal solo stato corrente essa sarà situata all'interno del CASE-WHEN di ogni singolo stato senza condizioni riguardanti gli ingressi.

Testbench: test_gest_cassaforte.vhd

```
28  LIBRARY ieee;
29  USE ieee.std_logic_1164.ALL;
30  USE ieee.std_logic_arith.ALL;
31  -- Uncomment the following library declaration if using
32  -- arithmetic functions with Signed or Unsigned values
33  --USE ieee.numeric_std.ALL;
34
35  ENTITY test_gest_cassaforte IS
36  END test_gest_cassaforte;
37
38  ARCHITECTURE behavior OF test_gest_cassaforte IS
39
40      -- Component Declaration for the Unit Under Test (UUT)
41
42      COMPONENT gest_cassaforte
43      PORT(
44           clk : IN  std_logic;
45           rst : IN  std_logic;
46           ric1 : IN  std_logic_vector(0 to 1);
47           ric2 : IN  std_logic_vector(0 to 1);
48           sens_porta : IN  std_logic;
49           timeout : IN  std_logic;
50           apertura : OUT  std_logic;
51           disp : OUT  unsigned(0 to 3);
52           En_cont : OUT  std_logic
53          );
54      END COMPONENT;
55
56
57     --Inputs
58     signal clk : std_logic := '0';
59     signal rst : std_logic := '0';
60     signal ric1 : std_logic_vector(0 to 1) := (others => '0');
61     signal ric2 : std_logic_vector(0 to 1) := (others => '0');
62     signal sens_porta : std_logic := '0';
63     signal timeout : std_logic := '0';
64     signal temp_in : std_logic_vector(0 to 5);
65     --Outputs
66     signal apertura : std_logic;
67     signal disp : unsigned(0 to 3);
68     signal En_cont : std_logic;
69
70     -- Clock period definitions
71     constant clk_period : time := 10 ns;
72
73     procedure test_START(signal temp_in : OUT std_logic_vector(0 to 5);
74                          signal rst : OUT std_logic) is
75     begin
76
77        rst<='1';
78        wait for clk_period*10;
79
80     -- test stato START
81        for i in 0 to 63 loop
82           temp_in<=conv_std_logic_vector(i,6);
83           rst<='0';
84           wait for 10 ns;
85           rst<='1';
86           wait for 20 ns;
87
88        end loop;
89     -- fine test stato START
90
91     end test_START;
92
93     procedure test_RIC_SEC(signal temp_in : OUT std_logic_vector(0 to 5);
94                          signal rst : OUT std_logic) is
95     begin
96
```

Abbiamo deciso di effettuare una testbench in grado di simulare l'andamento del modulo per tutte le possibili transazioni di stato per ogni stato.

Per rendere il più semplice e modulare possibile la simulazione effettiva abbiamo deciso di implementare all'interno del codice delle PROCEDURE in cui immettere i pezzi di codice che testano tutte le possibili combinazioni per ogni stato. Per la PROCEDURE TEST_START abbiamo un for che testa tutte le 64 possibili combinazioni, ogni volta prima di cambiare la

```
97        rst<='1';
98        wait for clk_period*10;
99
100
101     -- test stato RIC_SEC
102        for i in 0 to 63 loop
103           temp_in<="010000";
104           rst<='0';
105           wait for 10 ns;
106           temp_in<=conv_std_logic_vector(i,6);
107           wait for 10 ns;
108           rst<='1';
109           wait for 20 ns;
110        end loop;
111     -- fine test stato RIC_SEC
112
113     end test_RIC_SEC;
114
115     procedure test_ANOMALIA(signal temp_in : OUT std_logic_vector(0 to 5);
116                             signal rst : OUT std_logic) is
117     begin
118
119        rst<='1';
120        wait for clk_period*10;
121
122
123     -- test stato ANOMALIA
124        for i in 0 to 63 loop
125           temp_in<="000001";
126           rst<='0';
127           wait for 10 ns;
128           temp_in<=conv_std_logic_vector(i,6);
129           wait for 10 ns;
130           rst<='1';
131           wait for 20 ns;
132        end loop;
133     -- fine test stato ANOMALIA
134
135     end test_ANOMALIA;
136
137     procedure test_APRO(signal temp_in : OUT std_logic_vector(0 to 5);
138                         signal rst : OUT std_logic) is
139     begin
140
141        rst<='1';
142        wait for clk_period*10;
143
144     -- test stato APRO
145        for i in 0 to 63 loop
146           temp_in<="011000";
147           rst<='0';
148           wait for 10 ns;
149           temp_in<=conv_std_logic_vector(i,6);
150           wait for 10 ns;
151           rst<='1';
152           wait for 20 ns;
153        end loop;
154     -- fine test stato APRO
155
156     end test_APRO;
157
158     procedure test_APERTA(signal temp_in : OUT std_logic_vector(0 to 5);
159                           signal rst : OUT std_logic) is
160     begin
161
162        rst<='1';
163        wait for clk_period*10;
164
165
166     -- test stato APERTA
167        for i in 0 to 63 loop
168           temp_in<="011000";
169           rst<='0';
170           wait for 10 ns;
171           temp_in<="000001";
172           wait for 10 ns;
173           temp_in<=conv_std_logic_vector(i,6);
174           wait for 10 ns;
175           rst<='1';
176           wait for 20 ns;
177        end loop;
178     -- fine test stato APERTA
179
180     end test_APERTA;
181
182  BEGIN
183
184     -- Instantiate the Unit Under Test (UUT)
185     uut: gest_cassaforte PORT MAP (
186             clk => clk,
187             rst => rst,
188             ric1 => ric1,
189             ric2 => ric2,
190             sens_porta => sens_porta,
191             timeout => timeout,
192             apertura => apertura,
193             disp => disp,
194             En_cont => En_cont
195           );
196
197     -- Clock process definitions
198     clk_process :process
199     begin
```

combinazione in ingresso abbiamo sempre resettato il modulo.

Per le PROCEDURE TEST_RIC_SEC/ANOMALIA/APRO siccome ogni volta che resettiamo il modulo ripartiamo dallo stato START abbiamo dovuto prima immettere una combinazione “sicura” che ci facesse arrivare nello stato da testare per poi procedere col test delle singole combinazioni per quello stato.

Per la PROCEDURE TEST_APERTA non avendo una combinazione diretta che ci facesse passare dallo stato START ad APERTA abbiamo dovuto utilizzare 2 combinazioni “sicure”, la prima per poter arrivare allo stato APRO e la seconda per poter arrivare allo stato da testare.

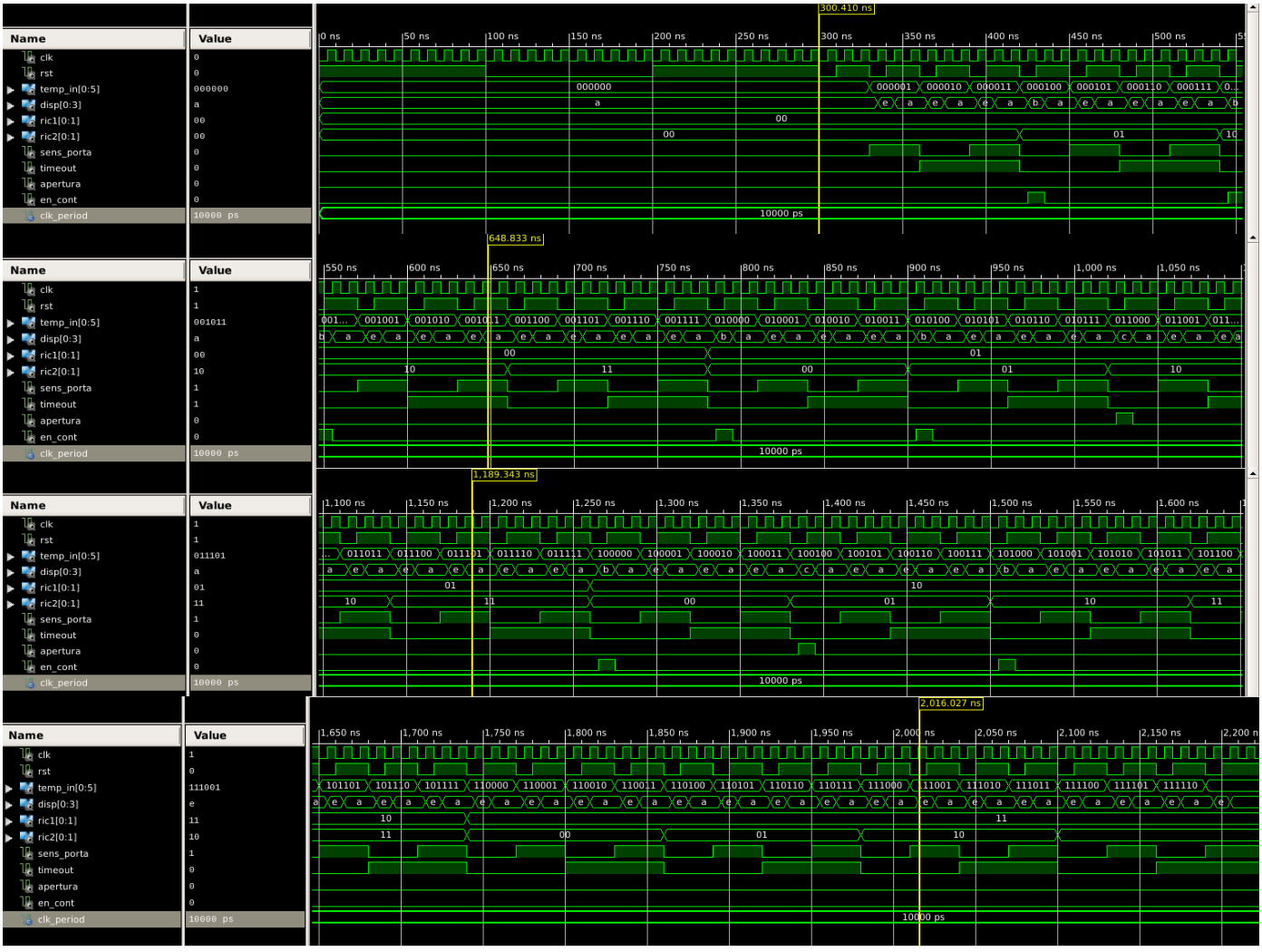
```
200         clk <= '0';
201         wait for clk_period/2;
202         clk <= '1';
203         wait for clk_period/2;
204     end process;
205
206
207     -- Stimulus process
208     stim_proc: process
209
210     begin
211         -- hold reset state for 100 ns.
212         temp_in<="000000";
213         rst<='1';
214         wait for 100 ns;
215         rst<='0';
216         wait for clk_period*10;
217
218         --per testare ogni singolo stato lasciare decommentato quello desiderato
219
220         test_START (temp_in,rst);
221         --test_RIC_SEC (temp_in,rst);
222         --test_ANOMALIA (temp_in,rst);
223         --test_APRO (temp_in,rst);
224         --test_APERTA (temp_in,rst);
225
226         wait;
227     end process;
228
229     ric1 <= temp_in(0) & temp_in(1);
230     ric2 <= temp_in(2) & temp_in(3);
231     timeout <= temp_in(4);
232     sens_porta <= temp_in(5);
233
234 END;
```

Nel process di test effettivo abbiamo implementato le singole procedure in modo da poter scegliere più facilmente quale andare a visualizzare in fase di simulazione. Per decidere quale visualizzare basta commentare/decommentare le righe 220-224. Abbiamo preferito simulare ogni stato separatamente in modo da semplificare lo studio delle onde.

Per facilitarci l'utilizzo delle procedure all'interno della simulazione abbiamo utilizzato un vettore temporaneo temp_in, dichiarato in riga 64, per l'utilizzo dei for e per il passaggio dei valori tra procedure e process.

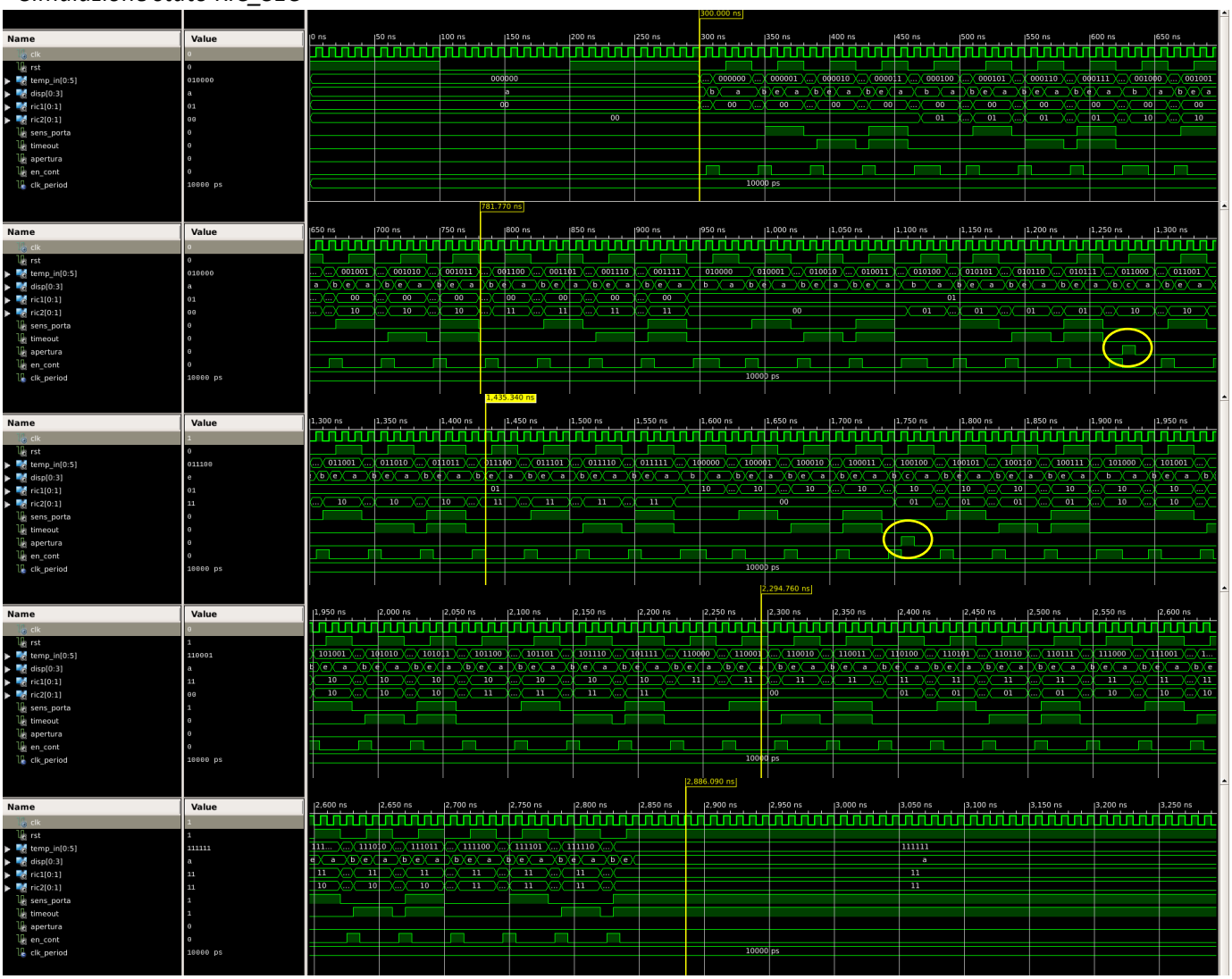
Al termine della testbench in modo concorrente il vettore temp_in viene risuddiviso ai corrispettivi ingressi. Questo vettore ci aiuta anche nella visualizzazione della combinazione in ingresso in fase di simulazione.

Simulazione stato START



Per fare un controllo di funzionamento ci siamo basati sui valori riportati dal display e sulle volte in cui l'uscita subiva una variazione ed abbiamo controllato che le combinazioni fossero esattamente quelle riportate nell'automa ogni volta che en_cont diventgava pari ad 1. Per lo stato di errore ci siamo riferiti solamente a ciò che veniva visualizzato sul display. Ciò fa capire l'effettiva utilità del display, che può fornirci lo stato corrente.

Simulazione stato RIC_SEC



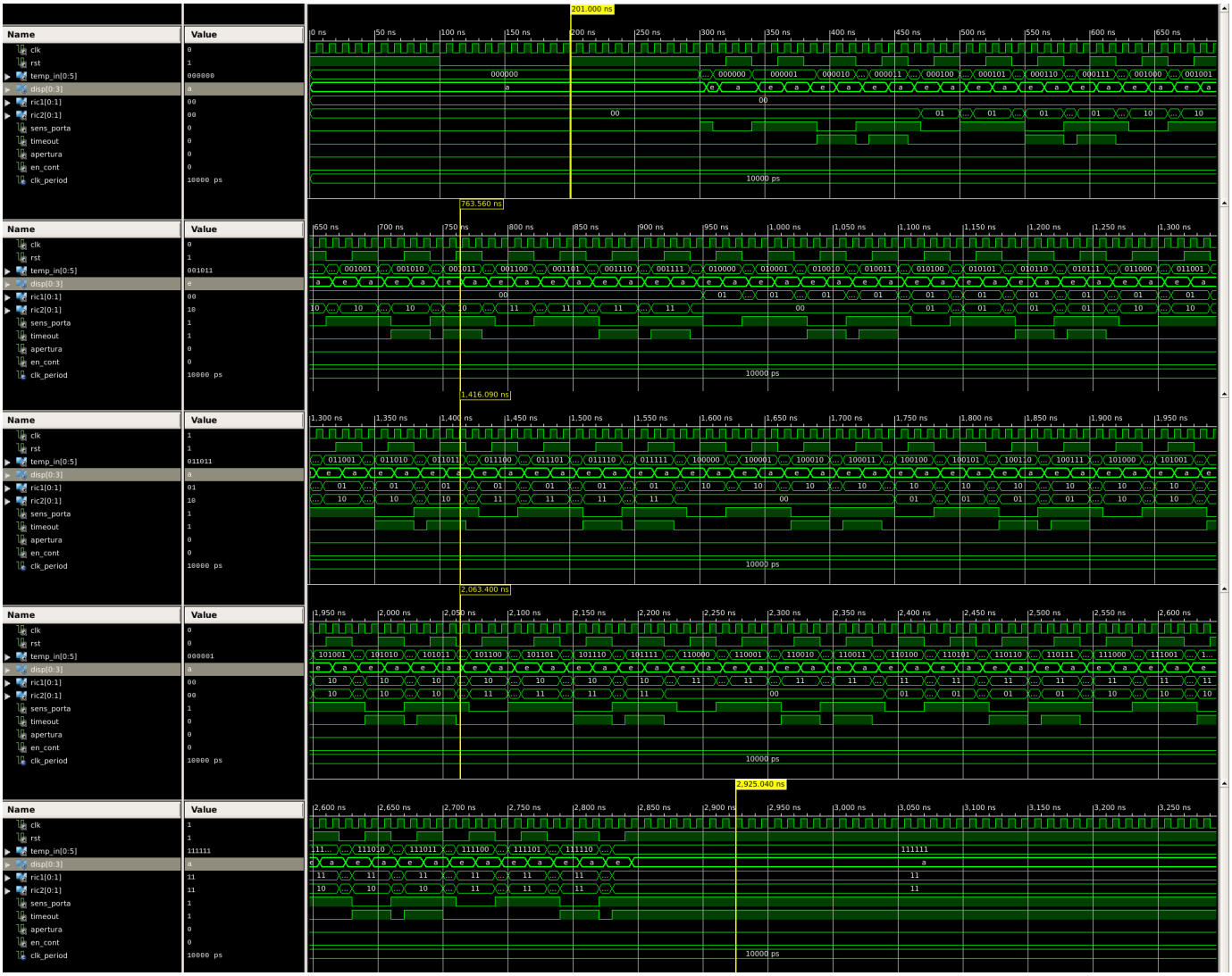
In questa simulazione prima di poter arrivare allo stato RIC_SEC possiamo notare il passaggio per lo stato START.

Come si può notare nella prima immagine grazie alla sequenza "000000" il display ci mostra un passaggio repentino dallo stato 'b' allo stato 'a'. Ciò vuol dire che con quella specifica combinazione avviene la transazione di stato indicata nell'automa di ritorno allo stato iniziale.

Nelle immagini 2 e 3 cerchiate in giallo ci sono le uscite "apertura" che indicano il passaggio di stato ad APRO tramite le combinazioni "011000" e "100100".

In tutte le altre combinazioni quando abbiamo una lunghezza d'onda maggiore per l'uscita en_cont vorrà dire che permaniamo nello stato, altrimenti quando la lunghezza è minore andiamo nello stato ANOMALIA.

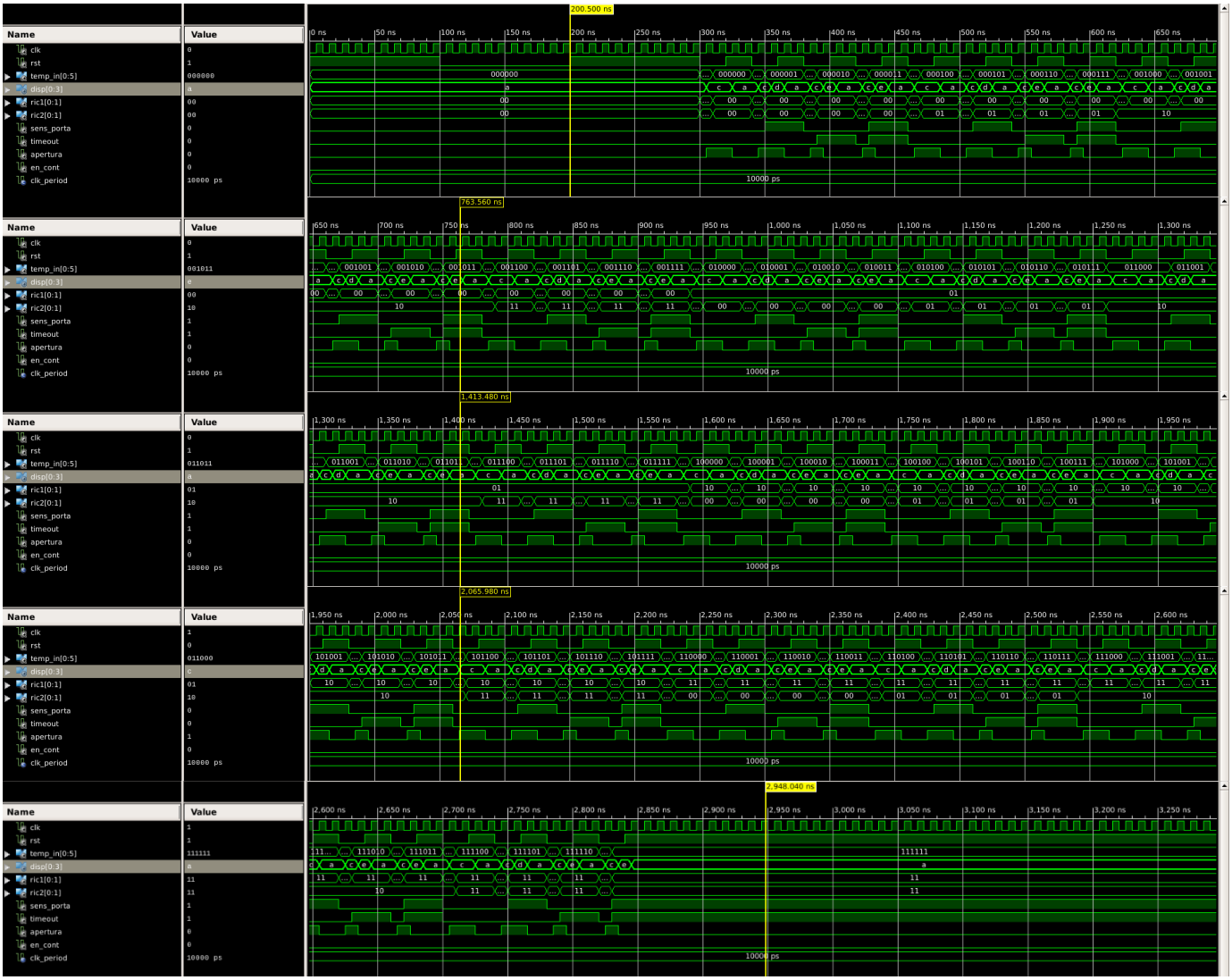
Simulazione stato ANOMALIA



In questa simulazione prima di poter arrivare allo stato ANOMALIA possiamo notare il passaggio per lo stato START.

Come si può notare nella prima immagine grazie alla sequenza “000000” il display ci mostra un passaggio repentino dallo stato ‘e’ allo stato ‘a’ mentre per tutte le altre combinazioni avviene una permanenza nello stato ‘e’ prima del reset dell’automa allo stato ‘a’. Ciò vuol dire che con quella specifica combinazione avviene la transazione di stato indicata nell’automa.

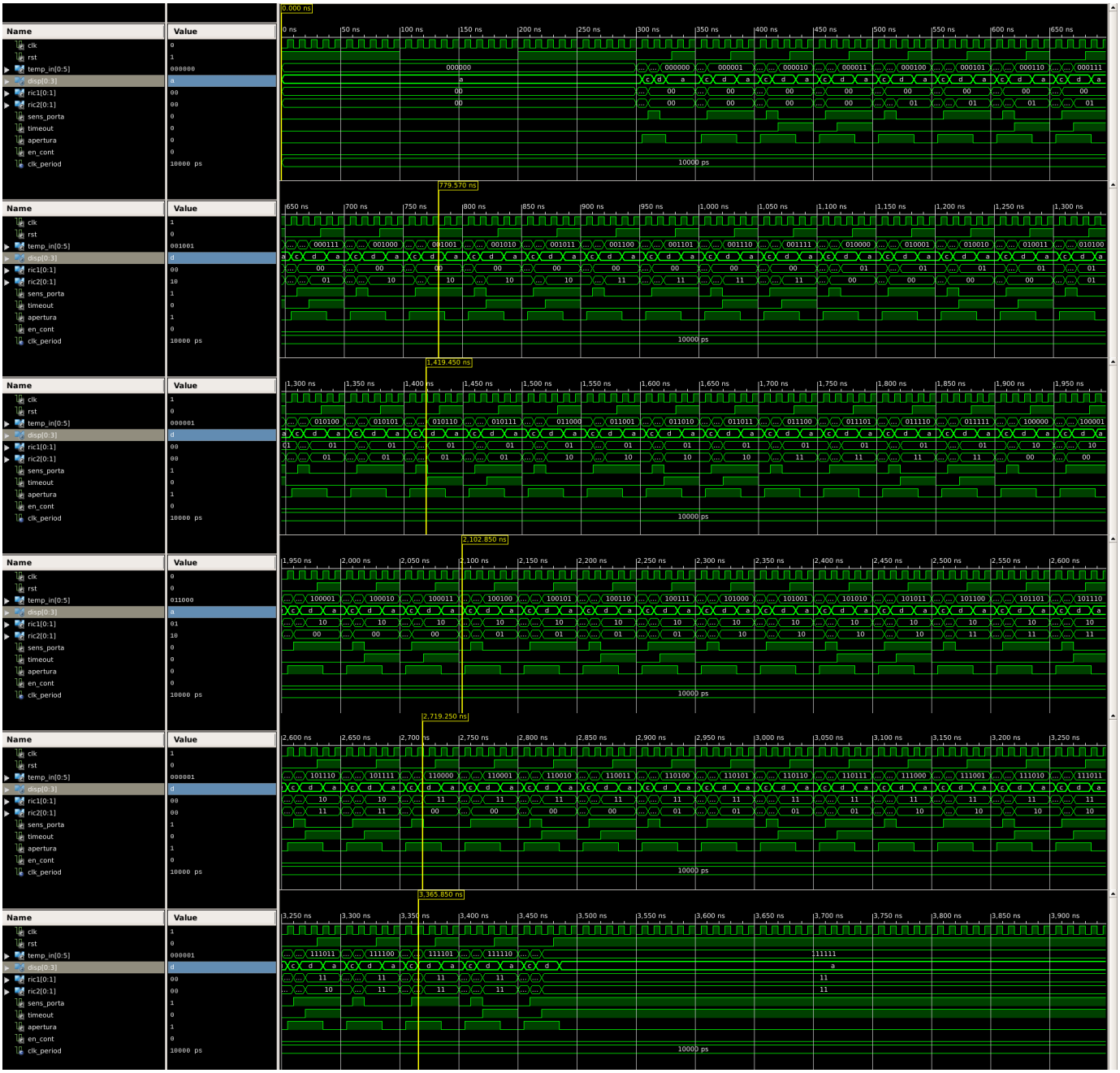
Simulazione stato APRO



In questa simulazione prima di poter arrivare allo stato APRO possiamo notare il passaggio per lo stato START.

In questa simulazione osservando l'uscita apertura notiamo una differenza di lunghezza d'onda che indica il passaggio dallo stato APRO allo stato ANOMALIA quando l'onda è più corta mentre indica o la permanenza nello stato o il passaggio ad APERTA quando è più lunga. Per una più precisa analisi basta osservare come si comporta il display che ci mostra lo stato corrente in ogni posizione.

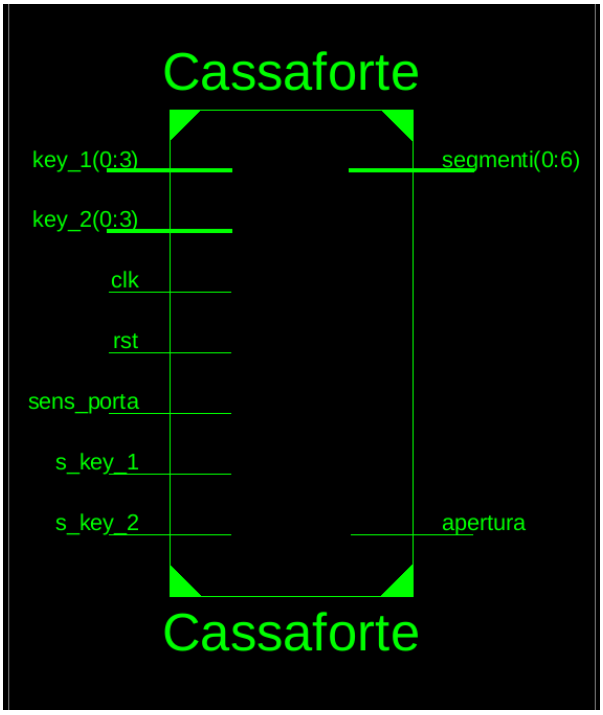
Simulazione stato APERTA



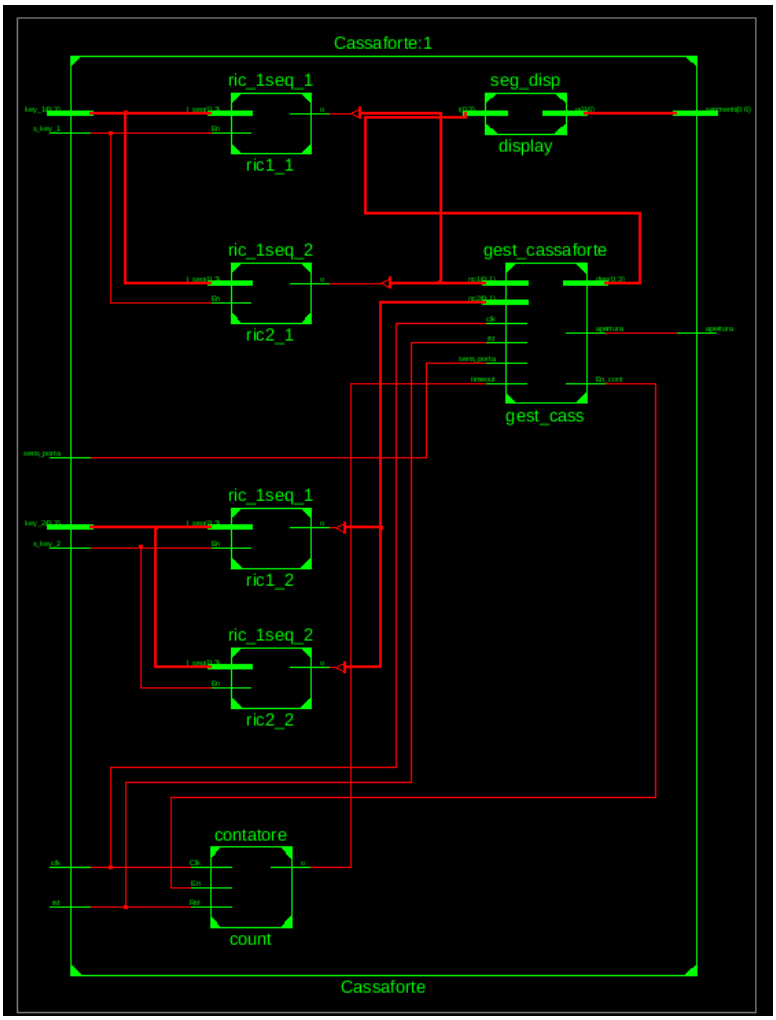
In questa simulazione prima di poter arrivare allo stato APERTA possiamo notare il passaggio per lo stato START ed APRO proprio perché non vi è combinazione diretta dallo stato iniziale.

Come si può notare nella prima immagine grazie alla sequenza “000000” il display ci mostra un passaggio repentino dallo stato ‘d’ allo stato ‘a’ mentre per tutte le altre combinazioni avviene una permanenza nello stato ‘d’ prima del reset dell’automa allo stato ‘a’. Ciò vuol dire che con quella specifica combinazione avviene la transazione di stato indicata nell’automa.

➢ *TOP MODULE* *(torna su)*



Entity principale



RTL-Schematic del top-module

Il top-module ci permette di interconnettere tramite una vista Structural tutti i componenti visti in precedenza in modo da realizzare il funzionamento della cassaforte descritto all'inizio.

Il nostro controllore possiede come ingressi:

- key_1 e key_2 che sono 2 bus (vettori) di lunghezza 4 rappresentanti le sequenze generate dalle chiavi da immettere nella cassaforte per poter provare ad iniziare il procedimento di apertura
- s_key_1 e s_key_2 che sono segnali rappresentanti i sensori che rivelano se la chiave è correttamente inserita nella serratura
- sens_porta che è il segnale rappresentante il sensore che indica se la porta della cassaforte è completamente chiusa o meno
- clk che è il è il segnale rappresentante il clock su cui si sincronizza tutta la macchina
- rst che è il segnale che resetta tutta la macchina

Il nostro controllore possiede come uscite:

- segmenti che è un display a 7 segmenti rappresentato da un bus (vettore) di lunghezza 7 che abbiamo deciso di implementare per facilitare la visualizzazione degli stati su cui si basa la macchina
- apertura che è il segnale rappresentante lo sblocco o meno della porta della cassaforte

Codice: Cassaforte.vhd

```vhdl
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.std_logic_arith.ALL;

-- Uncomment the following library declaration if using
-- arithmetic functions with Signed or Unsigned values
--use IEEE.NUMERIC_STD.ALL;

-- Uncomment the following library declaration if instantiating
-- any Xilinx primitives in this code.
--library UNISIM;
--use UNISIM.VComponents.all;

entity Cassaforte is
Generic(
   seq1 : std_logic_vector(0 to 3) :="0011";    --sequenze di default da riconoscere, ma che possono
   seq2 : std_logic_vector(0 to 3) :="1000"     --essere definite in fase di test
);
Port(
      clk,rst : in std_logic;
      key_1 : in std_logic_vector(0 to 3);
      key_2 : in std_logic_vector(0 to 3);
      s_key_1,s_key_2,sens_porta: in std_logic;

      apertura : out std_logic;
      segmenti: out std_logic_vector(0 to 6)
);
end Cassaforte;

architecture Structural of Cassaforte is
--definizione dell'entity del contatore
   component contatore is
   Generic(
      n : integer := 10
   );
   Port(
      Clk,Rst : in std_logic;
      En : in std_logic;

      o : out std_logic

   );
   end component;

   component ric_1seq is
      Generic(
         seq : std_logic_vector(0 to 3) :="0000"
      );
   Port(
         En : in std_logic;
         I_seq : in std_logic_vector(0 to 3);

         o : out std_logic
   );
   end component;

   component seg_disp is
   port
   (
      i :in unsigned(0 to 3);
      o :out STD_Logic_vector(0 to 6)
   );
   end component;

   component gest_cassaforte is
   Port(
      clk,rst : in std_logic;
      ric1 : in std_logic_vector(0 to 1);
      ric2 : in std_logic_vector(0 to 1);
     sens_porta: in std_logic;
     timeout: in std_logic;

     apertura : out std_logic;
     disp: out unsigned(0 to 3);
     En_cont: out std_logic
   );
   end component;

   signal timeout,En_cont : std_logic;
   signal Ric1,Ric2 :std_logic_vector(0 to 1);
   signal disp: unsigned(0 to 3);

begin
   --istanziazione del contatore che fa da timer
   count: contatore generic map(n=>10) port map(clk,rst,En_cont,timeout);
   --istanziazione dei riconoscitore per la prima chiave
   --il primo per riconoscere la 1° sequenza, il 2° per riconoscere la seconda
   ric1_1: ric_1seq generic map(seq=>seq1) port map(s_key_1,key_1,ric1(0));
   ric2_1: ric_1seq generic map(seq=>seq2) port map(s_key_1,key_1,ric1(1));
   --istanziazione dei riconoscitore per la seconda chiave
   --il primo per riconoscere la 1° sequenza, il 2° per riconoscere la seconda
   ric1_2: ric_1seq generic map(seq=>seq1) port map(s_key_2,key_2,ric2(0));
   ric2_2: ric_1seq generic map(seq=>seq2) port map(s_key_2,key_2,ric2(1));
   --istanziazione del gestore cassaforte (automa)
   gest_cass: gest_cassaforte port map(clk,rst,ric1,ric2,sens_porta,timeout,apertura,disp,En_cont);
   --istanziazione del controller per il display a 7 segmenti
   display: seg_disp port map(disp,segmenti);

end Structural;
```

Osservando il codice si può notare che abbiamo istanziato 4 volte il riconoscitore di sequenza, 2 per ogni chiave. Ciò è servito per permettere di riconoscere entrambe le sequenze per ogni chiave in modo da renderle invertibili a proprio piacimento purché le sequenze siano diverse tra loro.

Per ogni chiave istanziamo 2 riconoscitori in modo tale che il primo riconosca la prima sequenza (seq1) ed il secondo la seconda (seq2) ognuna mappata singolarmente nel generic opportuno. Il mapping della porta, invece, prevede che l'enable (En) venga collegato al sensore della chiave s_key1 o s_key2 in modo tale che il componente inizi a riconoscere la sequenza solamente quando è stata correttamente inserita una chiave, le sequenze in input saranno key1 o key2 ossia le sequenze lette in input dal controllore. Le uscite, inoltre, viste in coppia saranno concatenate all'interno di un vettore di 2 bit Ric1 o Ric2 che a loro volta saranno dati in ingresso al gest_cassaforte (automa).

Per il timeout del nostro automa abbiamo istanziato un contatore che riceve come generic il numero di clock da contare (10 nel nostro caso). Questo componente essendo sequenziale ha bisogno di un clock e di un reset con cui sincronizzarsi ed in questo caso saranno proprio il clock ed il reset dell'entity principale. Il contatore per essere abilitato avrà bisogno di un enable che sarà proprio quello generato dall'automa (En_cont). L'uscita di questo modulo sarà proprio il segnale di timeout che verrà dato come input proprio all'automa.

Display è l'istanziazione del controller per il display a 7 segmenti al quale mappiamo in input il numero unsigned (disp) da rappresentare a schermo, dato in output dall'automa (gest_cassaforte), e restituisce in output l'effettiva codifica "abcdefg" per la rappresentazione che andrà collegata in uscita (segmenti) al nostro controllore.

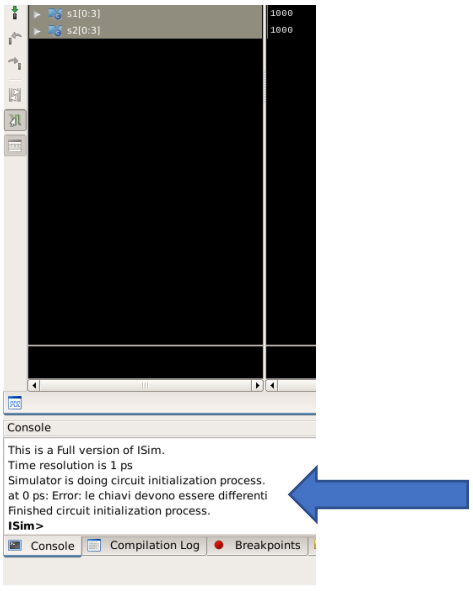
L'ultimo modulo istanziato corrisponde proprio all'automa (gest_cassaforte) che avrà in input sicuramente il clock (clk) ed il reset (rst) per sincronizzarsi con tutta la macchina, i vettori che concatenano le uscite dei riconoscitori di sequenza (ric1 e ric2), il sensore della porta per verificarne la chiusura o meno (sens_porta) ed il timeout (timeout) generato dal timer (contatore). Talvolta genererà come output il segnale di apertura o chiusura della serratura della nostra cassaforte (apertura), il numero/lettera da visualizzare sul display (disp) ed infine il segnale per abilitare o disabilitare il contatore (En_cont).

Testbench: test_cass_finale.vhd

```
LIBRARY ieee;
USE ieee.std_logic_1164.ALL;

-- Uncomment the following library declaration if using
-- arithmetic functions with Signed or Unsigned values
--USE ieee.numeric_std.ALL;

ENTITY test_cass_finale IS
END test_cass_finale;

ARCHITECTURE behavior OF test_cass_finale IS

    -- Component Declaration for the Unit Under Test (UUT)

   COMPONENT Cassaforte
    Generic(
      seq1 : std_logic_vector(0 to 3);
      seq2 : std_logic_vector(0 to 3)
      );
    PORT(
         clk : IN  std_logic;
         rst : IN  std_logic;
         key_1 : IN  std_logic_vector(0 to 3);
         key_2 : IN  std_logic_vector(0 to 3);
         s_key_1 : IN  std_logic;
         s_key_2 : IN  std_logic;
         sens_porta : IN  std_logic;
         apertura : OUT  std_logic;
         segmenti : OUT std_logic_vector(0 to 6)
        );
   END COMPONENT;


   --Inputs
   signal clk : std_logic := '0';
   signal rst : std_logic := '0';
   signal key_1 : std_logic_vector(0 to 3) := (others => '0');
   signal key_2 : std_logic_vector(0 to 3) := (others => '0');
   signal s_key_1 : std_logic := '0';
   signal s_key_2 : std_logic := '0';
   signal sens_porta : std_logic := '0';

   --Outputs
   signal apertura : std_logic;
   signal segmenti : std_logic_vector(0 to 6);

   -- Clock period definitions
   constant clk_period : time := 10 ns;

   -- costanti che ci permettono di definire le chiavi da riconoscere
   constant s1 : std_logic_vector(0 to 3):="0010";
   constant s2 : std_logic_vector(0 to 3):="1000";
BEGIN

   -- Instantiate the Unit Under Test (UUT)

   uut: Cassaforte
   GENERIC MAP(
         seq1=>s1,
         seq2=>s2
      )
   PORT MAP (
          clk => clk,
          rst => rst,
          key_1 => key_1,
          key_2 => key_2,
          s_key_1 => s_key_1,
          s_key_2 => s_key_2,
          sens_porta => sens_porta,
          apertura => apertura,
          segmenti => segmenti
        );

   -- Clock process definitions
   clk_process :process
   begin
      clk <= '0';
      wait for clk_period/2;
      clk <= '1';
      wait for clk_period/2;
   end process;


   -- Stimulus process
   stim_proc: process
   begin

      --ASSERT PER FAR SÌ CHE LE 2 CHIAVI SIANO OBBLIGATORIAMENTE DIFFERENTI
      assert s1/=s2 report "le chiavi devono essere differenti" severity error;

      -- hold reset state for 100 ns.
      rst<='1';
      wait for 100 ns;
      rst<='0';
      wait for clk_period*10;

      --sequenza START-RIC_SEC-APRO-APERTA-START
      s_key_1<='1';
      key_1<="1000";
      wait for 10 ns;
      s_key_2<='1';
      key_2<="0010";
      wait for 10 ns;
      sens_porta<='1';
      wait for 10 ns;
      s_key_1<='0';
      s_key_2<='0';
      wait for 10 ns;
      sens_porta<='0';
      wait for 10 ns;
      ----------------------------------------
--fase di reset
      rst<='1';
      s_key_1<='0';
      key_1<="0000";
      s_key_2<='0';
      key_2<="0000";
      sens_porta<='0';
      wait for 100 ns;
      rst<='0';
      wait for clk_period*10;
      -- sequenza START-RIC_SEC(dopo 10 cicli di clock)-ANOMALIA-START
      s_key_1<='1';
      key_1<="1000";
      wait for 120 ns;
      s_key_2<='1';
      key_2<="0010";
      wait for 10 ns;
      s_key_1<='0';
      s_key_2<='0';

      wait;
   end process;

END;
```

Come si può notare in riga 78-79 abbiamo dichiarato 2 costanti contenenti le 2 sequenze ammissibili che garantiscono l'apertura della cassaforte. Queste, infatti, vanno mappate nel generic dell'entity Cassaforte.

Per far sì che le 2 sequenze ammissibili non possano essere uguali abbiamo definito un assert che genererà un messaggio di errore in caso di uguaglianza tra seq1 e seq2.



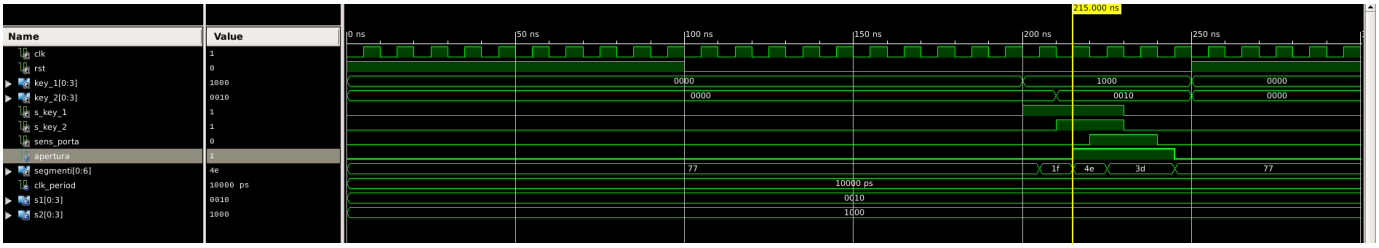
In questo testbench abbiamo simulato solamente 2 possibili percorsi dell'automa in modo da testare per bene i collegamenti tra i vari moduli poiché nell'effettivo l'automa è già stato completamente testato tramite il testbench dedicato come anche il resto dei moduli.

Abbiamo optato per non inserire i percorsi in procedure poiché non eccessivamente complicati da studiare insieme.

I due percorsi scelti sono:

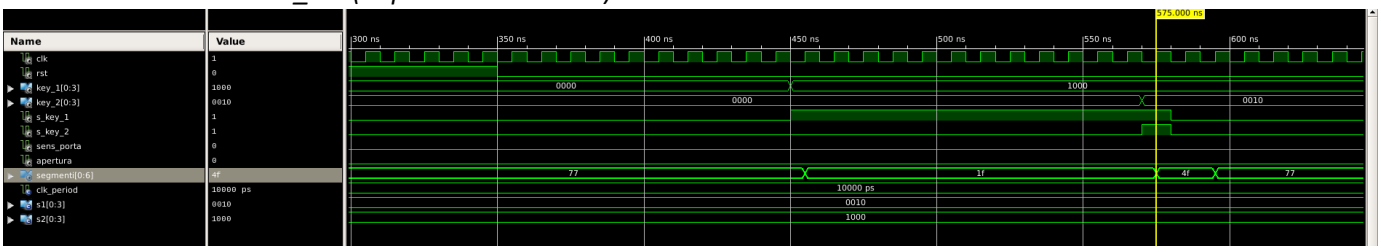
- START-RIC_SEC-APRO-APERTA-START: che testa il percorso principale ossia quello in cui avviene l'inserimento delle 2 chiavi corrette nel tempo limite e quindi la cassaforte verrà aperta ed infine chiusa fino a far tornare il sistema allo stato iniziale.
- START-RIC_SEC(dopo 10 cicli di clock)-ANOMALIA-START: che testa uno dei percorsi anomali in cui solo una chiave viene riconosciuta mentre la seconda chiave non viene inserita entro il tempo limite dei 10 cicli di clock e quindi si va in stato di anomalia fino poi a tornare allo stato iniziale ponendo tutti gli ingressi a 0.

Simulazione START-RIC_SEC-APRO-APERTA-START



Come si può notare dopo aver inserito la prima chiave correttamente ed anche la seconda prima che il timer dei 10 cicli di clock sia scaduto la nostra cassaforte si sblocca e può essere aperta, dopo averla aperta e richiusa avendo estratto entrambe le chiavi possiamo vedere come la macchina si riporta allo stato iniziale.

Simulazione START-RIC_SEC(dopo 10 cicli di clock)-ANOMALIA-START



Possiamo notare che dopo aver riconosciuto la prima chiave, avendo inserito la seconda dopo un tempo superiore ai 10 cicli di clock la macchina si porta allo stato di anomalia e lo capiamo grazie all'uscita segmenti che mostra "4f" cioè la lettera E. una volta rimosse le chiavi si vede che la macchina si riporta allo stato iniziale.

DESIGN SUMMARY REPORT *(torna su)*

Cassaforte Project Status (01/25/2020 - 00:59:23)			
Project File:	cassaforte.xise	**Parser Errors:**	No Errors
Module Name:	Cassaforte	**Implementation State:**	Synthesized
Target Device:	xc6slx4-3csg225	• **Errors:**	No Errors
Product Version:	ISE 14.7	• **Warnings:**	No Warnings
Design Goal:	Balanced	• **Routing Results:**	
Design Strategy:	Xilinx Default (unlocked)	• **Timing Constraints:**	
Environment:	System Settings	• **Final Timing Score:**	

Device Utilization Summary (estimated values)			[-]
Logic Utilization	**Used**	**Available**	**Utilization**
Number of Slice Registers	8	4800	0%
Number of Slice LUTs	25	2400	1%
Number of fully used LUT-FF pairs	8	25	32%
Number of bonded IOBs	21	132	15%
Number of BUFG/BUFGCTRLs	1	16	6%

Scheda di sviluppo selezionata: Spartan 6.

```
    ----------------------------------------------------------------------
    | States             | 5                                             |
    | Transitions        | 15                                            |
    | Inputs             | 5                                             |
    | Outputs            | 6                                             |
    | Clock              | clk (rising_edge)                             |
    | Reset              | rst (positive)                                |
    | Reset type         | synchronous                                   |
    | Reset State        | start                                         |
    | Power Up State     | start                                         |
    | Encoding           | auto                                          |
    | Implementation     | LUT                                           |
    ----------------------------------------------------------------------
    Summary:
        inferred   1 Finite State Machine(s).
Unit <gest_cassaforte> synthesized.

Synthesizing Unit <seg_disp>.
    Related source file is "/home/ise/Xilinx_Project/cassafortefinale/table.vhd".
    Found 16x7-bit Read Only RAM for signal <o>
    Summary:
        inferred   1 RAM(s).
Unit <seg_disp> synthesized.
```

Dalle informazioni riportate dall'HDL Synthesis riguardo gest_cassaforte possiamo notare che sono stati rilevati 5 stati con 15 transizioni, 5 input e 6 output. Abbiamo un clock attivo sul fronte di salita, un reset di tipo sincrono, uno stato di reset e accensione pari a START e la codifica dei vari stati automatica. Tutte queste informazioni rispecchiano le nostre specifiche di progettazione come si può notare anche dal modulo di gest_cassaforte riportato in precedenza.

Il Summary ci riporta anche che il display a 7 segmenti viene implementato attraverso una memoria RAM proprio come ci viene mostrato dal RTL Schematic del suddetto modulo.

```
=========================================================================
*                        Low Level Synthesis                            *
=========================================================================
Analyzing FSM <MFsm> for best encoding.
Optimizing FSM <gest_cass/FSM_0> on signal <c_s[1:3]> with user encoding.
----------------------
 State    | Encoding
----------------------
 start    | 000
 ric_sec  | 001
 anomalia | 010
 apro     | 011
 aperta   | 100
----------------------
```

Si può notare come vengono automaticamente codificati i 5 stati dell'automa.

APPROFONDIMENTI *(torna su)*

Durante la realizzazione della nostra macchina avremmo potuto decidere di non sfruttare un approccio modulare, ciò avrebbe complicato di gran lunga la comprensione ed il test del codice siccome non avremmo potuto testare singole parti separatamente. Un approccio non modulare avrebbe anche causato non poche difficoltà nella risoluzione di eventuali errori e warning nel corso dell'elaborazione.

Non è stato testato, ma lo stesso automa utilizzato per gest_cassaforte lo si sarebbe potuto realizzare anche mediante l'automa di Mealy sfruttando il template appropriato. Inoltre, abbiamo preferito utilizzare un template procedurale rispetto a quello tabellare per la generazione del next state poiché sarebbe risultato complicato gestire una tabella di tali dimensioni contenente tutte le combinazioni.

Un problema che il nostro template avrebbe potuto generare sarebbe potuto essere quello della generazione di Latch indesiderati, ma con i giusti accorgimenti ciò non è avvenuto.

Abbiamo optato per l'utilizzo di sensori che rilevassero l'effettiva presenza delle chiavi all'interno delle serrature per evitare problemi riguardanti letture indesiderate di bit in ingresso che avrebbero potuto far aprire la cassaforte in maniera anomala.

La scelta di utilizzare un sensore che ci rilevasse se la porta della cassaforte fosse chiusa o meno è stata fatta per realizzare un metodo di chiusura il più efficace possibile.

Abbiamo optato per utilizzare la codifica effettuata dal sintetizzatore, ossia la best encoding, poiché in linea con le nostre supposizioni progettuali. Qualora si fosse preferita una codifica differente avremmo potuto modificarla.